Utilizing Embedded Semantics for User-Driven Design of Pervasive Environments

Ahmet Soylu[1], Felix Mödritscher[2], and Patrick De Causmaecker[1]

[1] K. U. Leuven, Department of Computer Science, CODeS, iTec, Kortrijk, Belgium
{Ahmet.Soylu,Patrick.DeCausmaecker}@kuleuven-kortrijk.be
[2] Vienna University of Economics and Business, Department of Information Systems, Vienna, Austria
felix.moedritscher@wu.ac.at

Abstract. The Web does not only offer an almost infinitive number of services and resources but can be also seen as a technology to combine different technological devices, like mobile phones, digital media solutions, intelligent household appliances, tablet PCs, and any other kind of computers, in order to create environments satisfying the need of users. However, due to the large amount of web resources and services as well as the variety and range of user needs, it is impossible to realize software solutions for all possible scenarios. In this paper, we present a user-driven approach towards designing and assembling pervasive environments, taking into consideration resources and services available on the Web and provided through computing devices. Based on semantics embedded in the web content, we explain the concept as well as important components of this user-driven environment design methodology and show a first prototype. Finally, the overall approach is critically discussed from the perspectives of programmers and web users on the basis of related work.

Keywords: Semantic Web, End-User Development, Web Programming, Embedded Semantics.

1 Introduction

Presently, the Web comprises a network of linked documents, primarily designed for humans [1]. Humans either directly access web resources (i.e. pages) through URLs to find information of their interest or use web forms and other gadgets to interact with the web applications. Additionally, machines use ad-hoc techniques to extract information from the web content and a separate access channel/facade, i.e. web services, to interact with other web applications. However two particular movements in today's web technology and computing are on the way to change this picture. On the one hand, the Semantic Web alleviates data access problems of the machines, in a similar manner of web services, through a distinct facade by means of XML, RDF etc., thereby creating a Web which has facades of access and interaction for both humans and machines. On the other hand, with the emergence of Pervasive Computing, the Web is not a closed virtual box of networked applications and information anymore. It is supposed to be a communication and application space, i.e. Web of

S. Sánchez-Alonso and I.N. Athanasiadis (Eds.): MTSR 2010, CCIS 108, pp. 63–77, 2010.

Things (WoT), and an information space, i.e. Web of Data (WoD), for the computing systems [2, 3]. In other words, it is the medium of the immersion so that computing can be situated into the real life.

Overall the Web in its current form contains a lot of valuable data and functionality provided through web applications built by programmers from companies and applied science. The end-user, i.e. the humans, are involved into this process, e.g. through HCI methodologies like usability engineering or participatory design. Considering the large amount of data and functionality in the Web as well as the different end-users and devices, it is not possible to realize (working and learning) environments for all. Trying to overcome "one size fits all" flaw, streams like adaptive technology (e.g. Adaptive Hypermedia by [4]) or end-user development [5] have emerged. Due to a general criticism on adaptive technologies (e.g. in [6]), this paper focuses on end-user development (EUD) and introduces an approach towards "User-driven Design of Pervasive Environments" (UDPE) which is based on REST and embedded semantics. Therefore, section 2 elaborates theoretical foundations and argues for EUD. Then, section 3 describes UDPE from perspective of programming experts. Furthermore, section 4 sketches EUD facilities on the basis of technology and literature review. Finally, section 5 discusses UDPE with respect to related work, before section 6 concludes the paper along with the future work.

2 Foundations, Limitations, and Possible Solutions

The Web is becoming ubiquitous, semantic and more functional by the emergence of the pervasive computing era [7, 8], web semantics, novel architectural styles, and new development approaches. Within the context of this paper, embedded semantics and REST are of interest. Embedded semantics aims at enhancing HTML documents with semantic annotations in order to create a machine readable Web. It uses the attribute system of the HTML for structuring the valuable information. The important technologies are; microformats, RDFa, and eRDF. REST [9] is an architectural style aiming at collecting the fundamental design principles that enable the great scalability, growth and success of the Web [10]. It is built around the resources and representations. Web services are considered as resources which are meaningful concepts addressed with unique URIs while a representation is a document representing the state of a resource. Every interaction can be considered as a call of a particular resource and the result of each interaction as a new state of that resource. RESTful web services treat HTTP as a semantically capable application protocol rather than just a transport protocol, which is the case e.g. for SOAP [11].

We suggest that the usage of the Web (i.e. read/access and interact) should be considered in twofold: (1) machine facade and (2) human facade of the Web. Although a conceptual distinction is admissible, from a representational perspective a uniform representation of the both facades are desirable in order to prevent inconsistencies, synchronization problems and development overheads. Considering the matter from information access point of view, uploading an external file dedicated to machine reading (e.g. RDF or XML) still remains forbiddingly complex [12], hence we advocate that use of embedded semantic technologies will provide a simpler solution for unifying both machine and human readable facades of the web content.

From interaction point of view, annotating interactional elements such as forms and links can partially provide a similar unified facade for the interactions. Such unification is complete if the web site is fully RESTful that is all possible interactions provided through the site are built upon a REST API. To achieve this, HTML and REST have to be separated whereby HTML is used for the presentational structure only and relevant calls are made through the REST API. A web site based on this principle does not necessarily need to expose all the elemental functionalities of the web application through the human interface, but through the API. However, most APIs are only described with text in HTML documents for the use of developers [13]. Therefore, it is necessary to provide machine readable annotations for such descriptions in order to facilitate tool support for the developers [13].

Briefly, we speculate on the following model of the human-machine usable Web: web applications shall have one facade of access and interaction that is every web site dedicated to human use also becomes a web service. Machines also access to the human usable facade of the web applications. However since the required interactional elements and the valuable information are annotated, machines simply extract the annotated information and use it. Once a machine initiates an interaction through a resource, results are returned in human readable form (i.e. HTML) where each element of the result list (i.e. response) is also annotated. In short, semantic annotations shall define the functional and meaningful aspects of the application where HTML defines the presentational structure of the application. Hence within the same physical representation two different facades of use are realized. Considering the integration of the devices, use of RESTful services is an appropriate and a simple solution. Although such devices might deliver their functionalities through human usable embedded web sites, they are primarily expected to provide RESTful APIs.

From the perspective of developing pervasive environments, the theoretical issues described formerly indicate that traditional software development methods are inappropriate or hindering for end-users. On the one hand, the Web offers a large quantity of resources and semantic relations which cannot be fully taken into consideration by the developers. On the other hand, it is impossible to design environments for the needs of every potential user. Therefore, new research streams tries to overcome the flaws of "*one size fits all*" solutions. Amongst others, adaptation technologies aim at changing the behavior of a computer system according to the characteristics of the end-user or the environment [14], implying that adaptation builds upon user/environmental states, adaptable objects, and adaptation rules. However, [6] criticize that it is impossible to create adaptation rules for all possible situations. Consequently we focus on novel development methods from software engineering and human-computer interaction which aim at including end-users, e.g. through a participatory design, or even at shifting development tasks to them.

The latter approach depicted e.g. by Lieberman and colleagues [5] is called end-user development (EUD) and tries to change systems from being "*easy to use*" to being "*easy to develop*". The spectrum of EUD reaches from parameterization and customization of programs up to active programming and source code modifications. Depending on the expertise of the end-users, EUD has to provide programming tools on different levels, reaching from a source code editor up to high-level facilities which completely hideaway the programming tasks. Furthermore, a EUD framework has to consider and foster the necessary hand-on skills and competences. Overall,

EUD is a valuable approach to counter the dynamics and complexity of socio-technical systems and satisfy the end-users' needs by simply empowering them to develop their environments from the artifacts given themselves.

3 User-Driven Design of Pervasive Environments (UDPE): Basic Concept and a Possible Scenario

Indeed, annotations of web-based information and interactions lead to a machine-human usable, more functional Web, as available data and interactions can be projected to an object-based distributed database and to a distributed programming framework respectively. It is reasonable to consider annotated information through a page in terms of custom or predefined data structures or data types. An example is depicted in Fig. 1 where a hCard microformat is projected to a predefined (since hCard vocabulary is given) data structure. Furthermore, it is possible to project each annotated HTML form or link available through a resource (i.e. page) to a method/function where parameters are the input elements of the HTML form.

```
<div class="vcard">
 <span class="fn">Ahmet Soylu</span>
 <div class="adr">
  <span class="locality">Kortrik</span>
   <span class="country-name"> Belgium
   </span>
 </div>
 <div class="tel">0032484742034</div>
</div>
```

```
vCard {
   String name;
   Address adr;
   String tel;
 }
 Address {
   String locality;
   String country-name;
 }
```

Fig. 1. Projection of embedded information to data structures (hCard microformat example)

We face custom or predefined data structures on the one hand and functions of the web applications available through its different pages on the other hand. On top of that, it is possible to design an infrastructure which allows programmers to select and orchestrate different functionalities of web applications and also to use data available through each application as input parameters to these functions. Indeed a web application is just another site (also a web service) built upon other web sites and services. We differentiate between elemental web sites/services representing the core functionalities of a single application and compound web sites/services (i.e. mashups) which are composed of other sites and services. Mashups [15] represent a development paradigm by which the Web can be used as a distributed database and a programming framework, requiring that the resources, i.e. data instances and functions, are available on the Web at development and run time. It is apparent that different devices will be connected to the Web and serve their data and functionalities to users and each other over the Web by the means of REST-based services. Therefore by following the mashup approach, in accordance to [16], it is possible to enable developers to program pervasive spaces [17].

Combining the mashup and EUD approach, *User-driven Design of Pervasive Environments (UDPE)* refers to the HCI paradigm shift from "*what the Web can do*" to

"*what the Web can do for humans*" [18]. UDPE is not restricted to web programmers only. In the sense of end-user development, it applies to extended scenarios by empowering all kind of end-users to 'develop' ubiquitous spaces through the Web. In accordance with end-user development, such a framework for user-driven environment design must provide facilities for programmers (e.g. a code editor) as well as inexperienced users (e.g. web-widgets). The overall frame of the UDPE is depicted in Fig. 2 assuming existence of embedded web servers or gateways coupled with the internal functions of available devices.

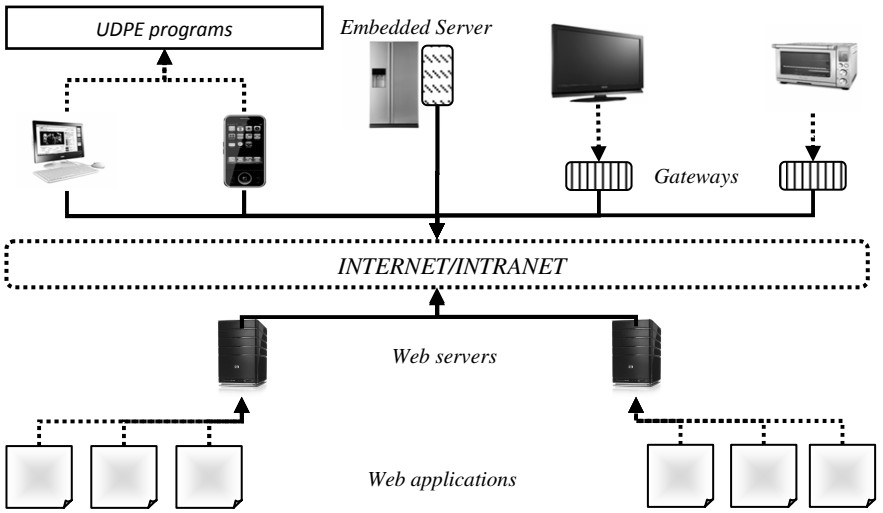

Fig. 2. User-driven design of pervasive environments through the mashup and EUD approach

A possible realization of UDPE based on REST and the embedded semantic technologies is described in the following. In a first sketch, we focus on the development of pervasive environments through experienced users (i.e. web programmers), then key features and UI elements for non-programmers will be addressed as well. In our scenario, producers of TVs and TV recorders include embedded servers into their new products. These embedded servers provide human usable web interfaces which publish the functionalities of the TVs and the recorders through RESTful APIs. For simplicity we assume existence of some basic functionalities; for TVs these are 'On', 'Off', 'Switch to channel A' etc., and for recorders these are 'On', 'Record', 'Off' etc.

The underlying implementation of the human user interface is based on the RESTful requests to the API. Hence for each action there is a corresponding REST request. It is assumed that these two devices are connected to a user's local network and to the Internet (e.g. through a local master server). The TV stations publish their daily schedules through a web site. We also assume that programs in the daily schedule are annotated with the hCalendar microformat. A portal site aggregates the schedules of the TV stations and provides a query form to the users so that they can query the TV schedules by providing day and TV station parameters. A developer has given a task

to program a web agent which allows user to set a recorder and a TV to record a particular TV show. Users only provide the name of the show, and/or the TV station, and then the agent is supposed to find out required time intervals through the station's web site in order to schedule the TV and the recorder.

At the first hand, we assume existence of the embedded web applications for TV and recorder and the semantic annotations of the TV programs in terms of RESTful requests and embedded semantics respectively. The RESTful requests together with their projections are given in Table 1 for a TV and recorder.

Table 1. Available RESTful requests for the TV and the recorder, and their projections

TV	**REST Request**	**Projected Method**
On/Off	PUT http://localhost/mytv/status/	put_status(String)
Switch	PUT http://localhost/mytv/channel/	put_channel(String)
Recorder	**REST Request**	**Projected Method**
On/Off	PUT http://localhost/myrec/status/	put_status(String)
Record	POST http://localhost/myrec/newRec/	post_newRecord(String)

The later step requires a detailed elaboration on a possible programming editor supporting UDPE. There are several requirements to be discussed around such an editor. The most fundamental requirement is existence of the content assistance. Content assistance, in our context, enables developer to navigate inside the function and data schema of a resource. For instance, when a developer wants to use data and functional sources of a particular site, after typing down the resource name (probably the variable name pointing out to the resource), the editor should fetch schemas of the available functions and data structures, and should display and recommend them as a selectable list in order to support the development process. Furthermore each element in the vocabulary of the schema should also have an accompanied human readable description. Although such a vocabulary is for machine use, such descriptors will support the human developer during the development process. This is crucial since in most of the cases developer might not be aware of the available functional and data sources through an application unless it is explicitly documented.

However, this feature is strictly based on the characteristics of the technology used to annotate the information and the interactions. Annotated information should exhibit 'locality' [19] and should have an accessible description. Descriptive characteristic might be inherent in the technology itself, that is, it might be 'self-contained' (e.g. RDFa) [19] or have a separate descriptor (e.g. eRDF). Locality requires a particular data structure within a HTML document to be accessible, while self-containment requires specific structures to be re-usable without dependency on any descriptor or pre-knowledge. If the embedded information exhibits locality and self-containment or refers to a separate descriptor, then the programming editor can access the page and retrieve schemas and descriptions of the embedded information and interactions.

The demonstration given in this paper is based on microformats [20] (but compliant with RDFa and eRDF). Although microformats exhibit locality, they do not fully

support self-containment as either the syntax and vocabulary must be given by the consumer agents or the resources are accompanied with extractors. None of these prerequisites satisfies our needs since our agent (i.e. editor) is expected to have no prior knowledge about the available microformat, and an extractor highly couples the agent and the underlying technology by extracting information only into a predefined format in a predefined way. Accordingly, we prefer to use a separate microformat descriptor used for specifying custom microformats. The data model of the descriptor includes the following required (r) and optional (o) fields: *(1) Type*: (r) Determines if it is an elemental or compound microformat. *(2) Identifier*: (r) Allows a differentiation between the elements. *(3) Design pattern*: (r) Specifies which HTML elements and attributes are used to define a certain microformat. *(4) Label*: (o) Provides a human readable description of the element. *(5) Match string:* (o) Restricts HTML attribute of the design pattern based on string equivalents or regular expressions. *(6) Scope*: (o) Specifies the scope of the semantics within the web content. *(7) Selector*: (o) Determines from which source the element text or semantic have to be extracted. *(8) Reference*: (o) Refers to another existing microformat, particularly useful for compound microformats. *(9) Optional:* (o) Indicates that an elemental microformat is optional within a given compound microformat.

This description language satisfies our needs for describing the schema of the information embedded in the web content. Referring interactions, we consider two forms to be annotated: RESTful APIs, and forms/links of web sites. We employ the microformat proposal hREST [13] which aims at providing machine readable descriptions of web-based APIs. Authors introduced six elements for the hREST microformat: (1) *service* as a main block markup to indicate that the corresponding microformat describes a service, (2) *operation* to annotate the service operations, (3) *address* which annotates the URI of the operation, (4) *method* to annotate the HTTP method (GET etc.), (5) *input and output* to annotate the input and output of an operation, and finally (6) *label* for human readable label of a service. Our microformat descriptor is suitable for describing the hREST microformat, so that, both together fully satisfy our requirements. Considering interactions, links represent one unified user action, for instance deleting a record or getting information of a particular resource. They do not require any input parameters since everything is predefined, they are self-descriptive, and they inherit self-containment. Therefore the only requirement, in our context, is existence of meaningful operation names and corresponding descriptions. We prefer to use the title attribute of the HTML link for declaring the name of the operation and an inner span element with the class value of 'nav_desc' to annotate the descriptions of the operations.

Forms are composed of meaningful elements, and their syntax and vocabulary are fixed thereby satisfying locality and self-containment, whereby the microformat community has already drafted a methodology for annotating forms (see http://microformats.org/wiki/rest/forms-brainstorming). The approach partially satisfies our requirements. The basic requirement deals with providing a label attribute for each input element which can be used for naming the input elements. However no means is introduced in order to shortly describe the form and the input elements which is a crucial requirement for a possible editor. Therefore we use inner span elements with the class attribute values of 'input_desc' and 'form_desc'. A possible

concern while using annotated forms or links is the response format. The response format (i.e. return type) is normally a HTML document which should but is not understandable for machine agents. In this context, the result format is still a HTML document where each result element is instance of a data chunk annotated within the HTML page and not necessarily having the same structure.

A crucial requirement for such an infrastructure is the availability of a generic event notifier - listener infrastructure in order to enable developers to set automatic actions associated with the events occurring (e.g. TV is on, a new record started etc.) in the environment. In this regard a context ontology, which is beyond the scope of this paper, with a rule layer supporting various types of rules, i.e. deduction, normative, reactive [21], can also be an appropriate choice for domain-specific applications. An example pseudo code demonstrating the UDPE, based on the scenario described, has been depicted in Fig. 3.

```
1   /* definitions */
2   Resource tv = "http://www.portal.com/schedule/";
3   Resource myTv = "http://localhost/mytv/";
4   Resource myRecorder = "http://localhost/myrec/";
5   Listener ls;
6   Time ts;
7   Time te;
8   /* find a particular movie and extract start and end times */
9   List tvProgram[] = tv.search("21.04.2010", "TV1");
10  for each item in tvProgram do
11    if item.Title = "The Cosby Show" then
12      ts = item.StartTime;
13      te = item.EndTime;
14      break;
15    end
16  end
17  /* turn devices on, record the programme, turn devices off */
18  if ts != null and te != null then
19    ls.HookAction(myTv.put_status("On"),time = ts);
20    ls.HookAction(myTv.put_channel("TV1"), time = ts);
21    ls.HookAction(myRecorder.post_newRecord("Cosby"),time = ts);
22    ls.HookAction(myRecorder.put_status("Off"), time = te);
23    ls.HookAction(myTv.put_status("Off"),time = te);
24  end
```

Fig. 3. An UDPE program for a smart home environment including a TV and a recorder

By defining the embedded semantics relevant for a scenario and utilizing the REST-based methods (indicated with the pseudo code in Fig. 3), web programmers can assemble their pervasive environment on top of existing data and functionality in the Web. Moreover, they even can combine the different devices for their everyday activities. In this variant, however, UDPE is only applicable for expert users, i.e. programmers in the field of web technologies. As suggested by EUD guidelines [22], UDPE should provide facilities for both expert users and novices. Thus the upcoming section proposes features and a user interface for end-users, i.e. users with appropriate knowledge about the Web.

4 End-User Facilities from Literature and Technology Review

It is impossible to anticipate all needs of the user within the broad context space of the ubiquitous applications. This fact justifies end-user involvement. We consider user-involvement in two means: (1) at development time (a) by user's being part of the development cycle through actively providing feedback on the design [23], (b) by enabling end-users to design and develop their own applications with high level tools (i.e. EUD) [24], (2) at run-time by enabling the user to intervene application's behaviors (a) directly (i.e. user-control) by deciding on appropriate behaviors [25], (b) indirectly (i.e. user mediation) by providing helpful feedback [26]. We consider the former, in terms of EUD, and the latter, in terms of user-control, as key paradigms to the success of Ubiquitous Computing. Although user-control and mediation approach is contradictory to the pervasive computing vision, which places an absolute focus on machine control, we argue that user-involvement at run-time, supported by adequate machine guidance and feedback mechanisms, is a must. This becomes apparent if one considers years of research a head probably only with a limited achievement towards the real machine intelligence [27].

In the frame of EUD, literature proposes two different line of approaches: (1) web design tools such as Microsoft FrontPage, Macromedia Dreamweaver etc. enable users to visually design and develop web pages and sites [28]; (2) mashup design and development tools such as Yahoo Pipes, IBM Mashup etc. support end-users in combining services and data from various sources to create new functionalities and content [29]. With the emergence of Web 2.0 applications, visual web design and development tools have lost their prominence for end-users, since various Web 2.0 applications allow creating readily operational applications without need of exhaustive design and configuration efforts. Moreover, mashups enable users to combine functionalities and data available on the Web thus increasing the reuse.

We believe that the essence of mashups depends on their ability to bridge the gap between end-users and the design and development of applications. Therefore, we have investigated several mashup design and development tools (listed in [29]), in terms of their end-user facilities, to develop a prototype mashup mockup supporting UDPE: (1) IBM Mashup Center, (2) Intel Mashmaker, (3) JackBe Presto, (4) Liquid Apps, (5) Open Mashup Studio, (6) Yahoo Pipes, and (7) Deri Pipes. We have derived the following criteria for a EUD-enabling GUI: (1) design facilities should follow real world mental models [30], (2) users should not be placed under a high cognitive load by means of overloaded forms and pages (e.g. configuration facilities) [31], (3) users should be confident that they have the full control and awareness of an application [25], and (4) the design should be engaging [32]. According to these criteria, the most appropriate design elements seem to be widgets, wires, drag and drop facilities, and intelligent guidance. We explain these elements through the prototype mockup in Fig. 4 representing a slightly different scenario.

The overall development environment follows a grocery-kitchen metaphor, i.e. find ingredients in the grocery and cook them in the kitchen. The upper part of the Fig. 4 (grocery) gives a persistent presentation of the available resources (i.e. devices and web services) and aims at creation of a permanent awareness and control of the resources. It enables users to select data, functionalities and event notifiers (by following

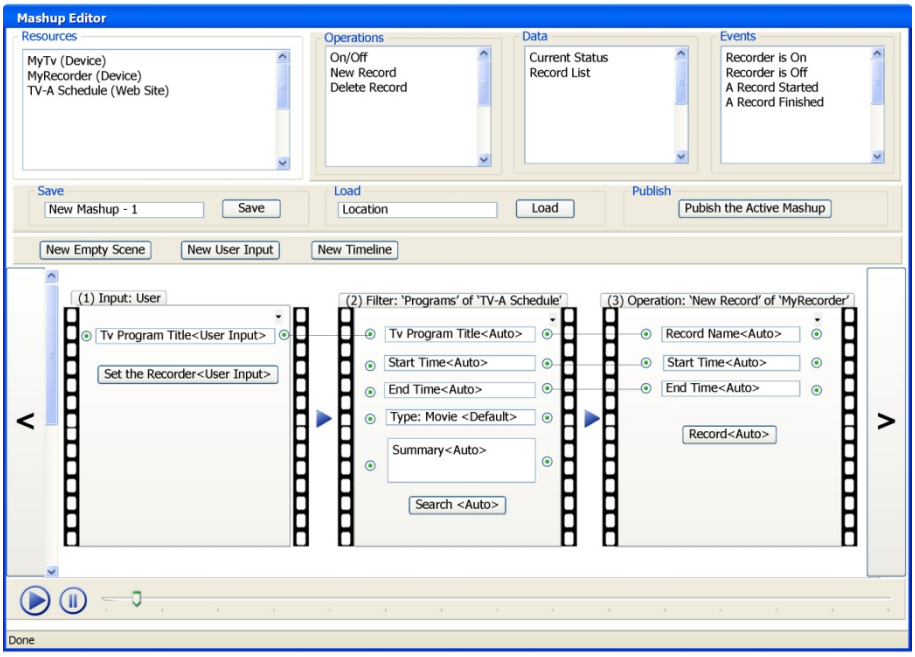

Fig. 4. A prototype mockup for UDPE Mashup Editor following grocery-kitchen and animation/movie metaphors

[33]) available through the devices and web applications (i.e. stores). Double clicking to a resource loads available elements of the selected resource into the respective containers (i.e. selection boxes for 'Operations', 'Data' and 'Events'). The bottom part of the Fig. 4 (kitchen) visualizes the mashup development area.

This development area follows a model similar to the animation/movie development environments (e.g. Microsoft Movie Maker - we have observed that many naive of users can create simple animations and movies through similar tools). It provides an execution timeline divided into scenes and aims at the creation of an engaging user experience. Each scene corresponds to a widget; there are four types of widgets available, namely, 'Operation', 'Filter', 'User', and 'Listener'. If user selects an operation from 'operation store' of the 'grocery', it is added as an 'operation' type widget (e.g. scene) into the timeline. Inputs of the operations are represented through visual input elements. If the user selects a 'data' (events, people, movies etc.) from the 'data store', a 'filter' type scene is added where each metadata element of the data type is displayed as an input element similar to an 'operation' widget. If the user selects an 'Event' from the 'store', a 'listener' type scene is added. It handles the event notifications sent by the resources (e.g. TV is on), and can be used to initiate execution of the mashup (being the first scene).

End-users can mark input elements as 'auto', 'user input', 'default', and 'fixed'. In case of 'auto' elements, input is provided by the widget wired to them, while 'user input' elements are filled by the user during execution of the mashup. 'Default' values are provided by the user at development time, whereby the user can change this value

at runtime. In the case of 'fixed' elements, the user enters a fix value at development time which cannot be changed in the execution phase. The mixture is also possible in a scene, that is, some input elements are filled by the user and some are auto completed from the previous scene. If there are no 'user input' elements in a scene, then the scene is not visible to the user during execution of the 'movie/animation'.

A user can create parallel timelines, so that she can wire different parallel data, operation, and listener widgets from other timelines to the widget at the next scene. Users can also change the order of widgets and wire widgets to each other through drag and drop facilities. The number of configuration elements is kept as minimal as possible, however the functionality can be increased through distributing different functionalities into layers (i.e. modes) according to their difficulties and anticipated usage frequencies ranging from 'beginner' to 'expert' levels. Intelligent guidance shall help through the design by means of disabling or selecting appropriate design elements in order to facilitate the design and development process.

5 Discussion and Related Work

So far we have not paid attention to the embedded semantics sufficiently, i.e. users should also have the possibility to specify information encoded in web-based content e.g. for describing RESTful services or for exchanging data between applications. Overall, the main focus of UDPE is set on Semantic Web technologies such as RDF and OWL. However we advocate that this simple idea of annotating information within HTML content introduces new possibilities like indicated in this paper. In [3], authors use embedded semantics, microformats, to annotate valuable information pieces and contextual information for the e-learning domain. Furthermore they built a web service which harvests embedded information and allows clients to query this information. A similar approach has been employed in [34] where microformats are used to find and annotate governmental web services. These services are harvested by special agents, and the description is stored in a semantic repository. Later these services are put at disposal of citizens by means of a semantic search engine.

Within the context of this paper, we also tried to build on existing mashup technologies [15, 35]. On the one hand the Mashup tools mentioned in section 4 have a strong focus on content aggregation and manipulation, i.e. feeds, while providing limited support for service composition. Microformats and RDFa are not supported and attention is given to feeds (e.g. RSS). Visual development environments are provided based on widgets, called modules or pipes, however support for more experienced users through source code manipulation is not well addressed. On the other hand, the underlying approach, technology and framework used in these tools are left hidden (most of them are already commercial, and not open source), therefore it is not possible to compare these approaches with ours from a technical point of view. A notable approach which is based on a concrete methodology and technology is SMashups [36]. It focuses on service composition rather than data. It follows the SAWSDL approach (Semantic Annotation for WSDL) which aims at adding semantic annotations to web services described with WSDL. A service annotation mechanism, called SA-REST, is based on Microformats [13] and RDFa [36] and used for REST based services usually described in HTML pages. SA-REST and SAWSDL specify

associations between the service description components and concepts in a semantic model (i.e. ontology) in order to enable semantic interoperability. The main drawback of this approach is that it assumes the existence of a (pre-defined) ontology so that different services can be semantically integrated. However such approach can be only useful for integrating a group of domain-specific applications since ontologies, by their nature, require commitment, which is hard to achieve even for flat vocabularies. The Web is highly heterogeneous. Therefore we prefer to avoid utilizing ontologies for such global integration purposes. The approach presented in this paper further extends the notion of mashups to the Pervasive Computing through integration of everyday devices to the Web and considering Web as an active programming framework rather than a passive information source (e.g. event notifiers - listeners).

Literature on Pervasive Computing outlines that RDF and OWL are basically used to formalize ontological models, although the current focus is mostly on developing generic but domain-specific context ontologies, see [37] for a review, to be used for reasoning purposes in order to provide an adaptive user experience, there is a lack of a generic framework ensuring a loose integration of the Web and the pervasive spaces. Use of ontologies as domain-specific artifacts, in pervasive spaces, can enhance the application of UDPE. Developers and end-users can really program the environment by making use of context ontologies of the pervasive spaces since such context ontologies will allow programmer to reach information about the entities available in the environment, their statuses, characteristics and relationships with each other. We believe that at the interface level (i.e. Web) use of metadata level semantics will be an appropriate and a simple solution by assuming that every system has its own local ontology and metadata schema. This is because it is always easier to define mappings between different metadata schemas than defining ontological mappings.

From the hardware point of view, implementation of RESTful services on an embedded system is realized in [38], authors also refer to the related literature, for instance, [39] attached a mini server ability to equipments such as to air-conditioners while [40] implemented an embedded temperature web controller. The current implementations of embedded web servers need to be standardized and negotiated with the appliance producers. Marching towards the vision of ubiquitous Internet, enabling web services on embedded systems is certainly on the to-do list of consumer electronics industry for the near future [38].

6 Conclusions and Further Work

In this paper, we have introduced a development methodology named User-driven Design of Pervasive Environments (UDPE). It aims at empowering users to program and design their ubiquitous spaces from the data and functionality available on the Web thereby considering the Web as a distributed database and programming framework integrated with the digital equipments available through the ubiquitous infrastructure. We have shown how UDPE can be set into practice on the basis of embedded semantics and RESTful web services and how pervasive environments can be developed by two important target groups, web programmers and web users.

Considerable work still remains to be carried out in this area, both in terms of further validation, and in more detailed exploration of the proposed approach. The

research direction can be itemized as follows: (1) realization of a prototypic implementation of the approach, (2) realization of a UDPE programming editor supporting GUI-based interface for end-users as well, (3) extending UDPE's technical frame by integrating ontological models of the pervasive environments, (4) investigation of standardized context dissemination mechanisms [8, 41], e.g. notifiers – listeners through push/pull, to realize automatic actions taken by the pervasive environments based on changes in the environment context (e.g. events), (5) investigation of standardized and generic security mechanisms for UDPE. Indeed the realization of the aforementioned research path is expected to result in generic and standardized web-based ubiquitous computing framework.

Acknowledgments. This paper is based on research funded by the Industrial Research Fund (IOF) and conducted within the IOF Knowledge platform "Harnessing collective intelligence in order to make e-learning environments adaptive" (IOF KP/07/006). Partially, it is also funded by the European Community's 7th Framework Programme (IST-FP7) under grant agreement no 231396 (ROLE project).

References

1. Ayers, D.: The Shortest Path to the Future Web. Internet Comput. 10, 76–79 (2006)
2. Soylu, A., De Causmaecker, P.: Merging Model Driven and Ontology Driven Development Approaches: Pervasive Computing Perspective. In: 24th International Symposium on Computer and Information Sciences (ISCIS 2009), pp. 730–735. IEEE Press, Los Alamitos (2009)
3. Soylu, A., De Causmaecker, P., Wild, F.: Ubiquitous Web for Ubiquitous Computing Environments: The Role of Embedded Semantics. J. Mob. Multimed. 6, 26–48 (2010)
4. Brusilovsky, P.: Adaptive Hypermedia. User Modeling and User-Adapted Interaction 11, 87–110 (2001)
5. Lieberman, H., Paterno, F., Klann, M., Wulf, V.: End-User Development: An Emerging Paradigm. In: Lieberman, H., Paterno, F., Wulf, V. (eds.) End-User Development. LNCS, vol. 4321, pp. 1–8. Springer, Heidelberg (2006)
6. Wild, F., Mödritscher, F., Sigurdarson, S.E.: Designing for Change: Mash-Up Personal Learning Environments. In: eLearning Papers (2008), ISSN: 1887-1542
7. Weiser, M.: The computer for the 21st century. Sci. Am., 94–98 (1991)
8. Soylu, A., De Causmaecker, P., Desmet, P.: Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering. J. Soft. 4, 992–1013 (2009)
9. Vinoski, S.: REST eye for the SOA guy. IEEE Internet Comput. 11, 82–84 (2007)
10. Riva, C., Laitkorpi, M.: Designing Web-Based Mobile Services with REST. In: Di Nitto, E., Ripeanu, M. (eds.) ICSOC 2007. LNCS, vol. 4907, pp. 439–450. Springer, Heidelberg (2009)
11. Dillon, T., Talevski, A., Potdar, V., Chang, E.: Web of Things as a Framework for Ubiquitous Intelligence and Computing. In: Zhang, D., Portmann, M., Tan, A.-H., Indulska, J. (eds.) UIC 2009. LNCS, vol. 5585, pp. 1–10. Springer, Heidelberg (2009)
12. Khare, R.: Microformats: the next (small) thing on the semantic Web? Internet Comput. 10, 68–75 (2006)

13. Kopecky, J., Gomadam, K., Vitvar, T.: hRESTS: An HTML Microformat for Describing RESTful Web Services. In: International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2008), pp. 619–625 (2008)
14. Mödritscher, F.: Adaptive E-Learning Environments: Theory, Practice, and Experience. Verlag Dr. Müller, Saarbrücken (2008), ISBN: 978-3-639-02635-1
15. Benslimane, D., Dustdar, S., Sheth, A.: Service Mashups: The New Generation of Web Applications. IEEE Internet Comput. 12, 13–15 (2008)
16. Mödritscher, F., Wild, F.: Personalized E-Learning through Environment Design and Collaborative Activities. In: Holzinger, A. (ed.) USAB 2007. LNCS, vol. 4799, pp. 377–390. Springer, Heidelberg (2007)
17. Helal, S.: Programming pervasive spaces. IEEE Pervasive Comput. 4, 84–87 (2005)
18. Mödritscher, F.: Semantic Lifecycles: Modelling, Application, Authoring, Mining, and Evaluation of Meaningful Data. Int. J. Knowl. Web Intell. 1, 110–124 (2009)
19. Adida, B.: hGRDDL: Bridging micorformats and RDFa. J. Web Semant. 6, 61–69 (2008)
20. Khare, R., Çelik, T.: Microformats: A pragmatic path to the Semantic Web. In: 15th International World Wide Web Conference, pp. 865–866 (2006)
21. Boley, H., Kifer, M., Patranjan, P.L., Polleres, A.: Rule interchange on the web. In: Antoniou, G., Aßmann, U., Baroglio, C., Decker, S., Henze, N., Patranjan, P.-L., Tolksdorf, R. (eds.) Reasoning Web. LNCS, vol. 4636, pp. 269–309. Springer, Heidelberg (2007)
22. Repenning, A., Ioannidou, A.: What Makes End-User Development Tick? 13 Design Guidelines. In: Lieberman, H., Paterno, F., Wulf, V. (eds.): End-User Development. LNCS, vol. 4321, pp. 51–86. Springer, Dordrecht (2006)
23. Begier, B.: Users' involvement may help respect social and ethical values and improve software quality. Inf. Syst. Frontiers, doi: 10.1007/s10796-009-9202-z
24. Fischer, G., Giaccardi, E.: Meta-Design: A Framework for the Future of End-User Development. In: Lieberman, H., Paterno, F., Wulf, V. (eds.): End-User Development. LNCS, vol. 4321, pp. 421–452. Springer, Dordrecht (2006)
25. Spiekermann, S.: User Control in Ubiquitous Computing: Design Alternatives and User Acceptance. Shaker Verlag, Aachen (2008)
26. Dey, A.K., Mankoff, J.: Designing mediation for context-aware applications. ACM Trans. Comput.-Hum. Interact. 12, 53–80 (2005)
27. Kasabov, N.: Evolving Intelligence in humans & machines: Integrative evolving connectionist systems approach. IEEE Comput. Intell. Mag. 3, 23–37 (2008)
28. Valderas, P., Pelechano, V., Pastor, O.: Towards an End-User Development Approach for Web Engineering Methods. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 528–543. Springer, Heidelberg (2006)
29. Taivalsaari, A.: Mashware: The future of web applications. Technical report, Sun Microsystems (2009)
30. Meadows, D.H.: Thinking is Systems. Chelsea Green Publishing (2008)
31. Hagras, H.: Embedding Computational Intelligence in Pervasive Spaces. IEEE Pervasive Comput. 6, 85–89 (2007)
32. O'Brien, H.L., Toms, E.G.: What is user engagement? A conceptual framework for defining user engagement with technology. J. Am. Soc. Inf. Sci. Technol. 59, 938–955 (2008)
33. Wild, F., Sigurðarson, E.S., Sobernig, S., Stahl, C., Soylu, A., Rebas, V., Górka, D., Danielewska-Tuecka, A., Tapiador, A.: An Interoperability Infrastructure for Distributed Feed Networks. In: 1st International Workshop on Collaborative Open Environments for Project-Centered Learning (COOPER 2007), Greece (2007)
34. Sabucedo, L.A., Rifón, L.A.: A Microformat Based Approach For Crawling And Locating Services In The E-government Domain. In: 24th International Symposium on Computer and Information Sciences, pp. 111–116. IEEE Press, Los Alamitos (2009)

35. Birman, K., Cantwell, J., Freedman, D., Huang, Q., Nikolov, P., Ostrowski, K.: Edge Mashups for Service-Oriented Collaboration. IEEE Comput. 42, 90–94 (2009)
36. Sheth, A.P., Gomadam, K., Lathem, J.: SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. IEEE Internet Comput. 11, 91–94 (2007)
37. Perttunen, M., Riekki, J., Lassila, O.: Context Representation and Reasoning in Pervasive Computing: a Review. Int. J. Multimed. Ubiquitous Eng. 4, 1–28 (2009)
38. Chang, C.E., Mohd-Yasin, F., Mustapha, A.K.: An implementation of embedded RESTful services. In: Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA 2009), pp. 45–50 (2009)
39. Lin, T., Zhao, H., Wang, J., Han, G., Wang, J.: An Embedded Web Server for Equipments. In: 7th International Symposium on Parallel Architectures, Algorithms and Networks, pp. 345–350. IEEE, Los Alamitos (2004)
40. Huang, J., Ou, J.Y., Wang, Y.: Embedded Temperature Web Controller Based on IPv4 and IPv6. In: 31st Annual Conf. of the IEEE Industrial Electronics Society, IECON 2005 (2005)
41. Gu, T., Pung, H.K., Zhang, D.Q.: A Service-Oriented Middleware for Building Context-Aware Services. J. Netw. Comput. Appl. 28, 1–18 (2005)